POLITECNICO
MILANO 1863

Code Inspection

version 1.0

5th February 2017

Authors:
Davide Moneta (808686)
Federico Oldani (806337)
Luca Oppedisano (878301)

Index

1 Purpose and scope

The purpose of this document is to provide the code inspection result of the open source software Apache OFBiz®.
Apache OFBiz® is an open source product for the automation of enterprise processes that includes framework components and business applications for ERP (Enterprise Resource Planning), CRM (Customer Relationship Management), E-Business / E-Commerce, SCM (Supply Chain Management), MRP (Manufacturing Resource Planning), MMS/EAM (Maintenance Management System/Enterprise Asset Management).

2 Classes assigned

The class assigned to our group is:
../apache-ofbiz-16.11.01/applications/marketing/src/main/java/org/apache/ofbiz/marketing/tracking/TrackingCodeEvents.java

3 Functional role of assigned set of classes

Our class has the task to manage events that involve TrackingCode, in the Marketing section of the application; an explanation of what a user can do with a TrackingCode can be found by looking at the OFBiz End-User Documentation here:

https://cwiki.apache.org/confluence/display/OFBENDUSER/Visual+Tracking+Codes

Analysing the code and reading the javadoc, when available, we managed to find out what are the events that can be managed in this class. The selected class offers the following methods to handle tracking codes:

- **checkTrackingCodeUrlParam:** Check if the tracking code (AutoTrackingCode) with id specified in the request is in the database (if there is no tracking code id in the request return *"success"*). If an exception is caught or it isn't in the database, return *"error"*, otherwise call the processTrackingCode method with trackingCode as a parameter.
- **checkPartnerTrackingCodeUrlParam:** Check if the tracking code (PartnerTrackingCode) with id specified in the request is in the database. If not, it takes the default tracking code in database, and saves the new tracking code in the data source. If no default tracking code is found, it creates an empty tracking code. If no tracking code id is in the request return *"success"*, if an exception is caught return *"error"*, otherwise call the processTrackingCode method with trackingCode as a parameter.

- **processTrackingCode**: reads cookies from a trackingCode, if the order have already gone into effect and have not expired, it writes the cookies into a response, saves in persistence a visit for this trackingCode and manages the session customization. If the trackingCode requested a forward/redirect, the response is redirected and null is returned, in all other cases *"success"* is returned.
- **checkTrackingCodeCookies**: it cycles through all the cookies of a request, skipping the orders that cannot be found or haven't gone into effect yet or have expired, for all the other it saves in persistence a visit, and at the end of the cycle it return *"success"*.
- **checkAccessTrackingCode:** this method checks if a valid ACCESS tracking code is contained in the *request* parameters. If so it returns "*success*" otherwise, after having written in the logs, it returns "*:_protect_:*".
- **removeAccesTrackingCodeCookie:** this method deletes ACCESS cookies found in *request* and adds them to *response*. Returns "*success*".
- **makeTrackingCodeOrders:** this method makes a list (of *GenericValue*) of one element which stores informations about the last occurrence of given types of cookie and, if it's valid, is returned.

4 List of issues found

4.1 Naming conventions

4.1.1 Constants are declared using all uppercase with words separated by an underscore

The following variable is a constant but it is not named all uppercase:

48	`public static final String module = TrackingCodeEvents .class.getName();`

The following table contains those variables and strings that we suggest to declare as constants, since they are used multiple times:

Used in line	**Code**
245	`int siteIdCookieAge = (60 * 60 * 24 * 365);`
248, 263, 448	`"Ofbiz.TKCD.SiteId"`
77, 169, 181, 185, 302, 360, 395, 430	`"success"`
66, 72, 103, 135, 162	`"error"`
60, 97, 174, 309, 364, 435	`"delegator"`
63, 100, 116, 142, 322, 384, 479	`"TrackingCode"`
63, 100, 116, 122, 143, 175, 192, 227, 237, 322, 346, 384, 479, 494	`"TrackingCodeId"`
123, 144, 227, 237, 390, 493	`"trackingCodeTypeId"`
192, 346	`"VisitId"`
179, 193, 335, 347, 392, 486	`"FromDate"`
183, 339, 393, 489	`"ThruDate"`
269, 453	`"Ofbiz.TKCD.UpdatedTimeStamp"`

4.2 Braces

4.2.1 All if, while, do-while, try-catch, and for statements that have only one statement to execute are surrounded by curly braces.

The following lines are not surrounded by curly braces and are not indented in a new line:

56	`if (UtilValidate .isEmpty (trackingCodeId )) trackingCodeId  = request .getParameter ("atc");`
180	`if (Debug .infoOn ()) Debug .logInfo ("The TrackingCode  with  ID [" + trackingCodeId  + "] has not yet gone into effect, ignoring this trackingCodeId" , module );`
184	`if (Debug .infoOn ()) Debug .logInfo ("The TrackingCode  with  ID [" + trackingCodeId  + "] has expired, ignoring this trackingCodeId" , module );`
228	`if (trackableLifetime .longValue () > 0) trackableCookie .setMaxAge (trackableLifetime .intValue ());`
230	`if (cookieDomain .length () > 0) trackableCookie .setDomain (cookieDomain );`
238	`if (billableLifetime .longValue () > 0) billableCookie .setMaxAge (billableLifetime .intValue ());`
240	`if (cookieDomain .length () > 0) billableCookie .setDomain (cookieDomain );`
266	`if (cookieDomain .length () > 0) siteIdCookie .setDomain (cookieDomain );`
272	`if (cookieDomain .length () > 0) updatedTimeStampCookie .setDomain (cookieDomain );`
281	`if (overrideLogo  != null) session .setAttribute ("overrideLogo" , overrideLogo );`
284	`if (overrideCss  != null) session .setAttribute ("overrideCss" , overrideCss );`
368	`if (UtilValidate .isEmpty (trackingCodeId )) trackingCodeId  = request .getParameter ("atc");`
397	`if (Debug .infoOn ()) Debug .logInfo ("The TrackingCode  with  ID [" + trackingCodeId  + "] has expired, ignoring this trackingCodeId" , module );`
402	`if (Debug .infoOn ()) Debug .logInfo ("The TrackingCode  with  ID [" + trackingCodeId  + "] has not yet gone into effect, ignoring this trackingCodeId" , module );`
487	`if (Debug .infoOn ()) Debug .logInfo ("The TrackingCode  with  ID [" + trackingCodeId  + "] has not yet gone into effect, ignoring this trackingCodeId" , module );`
490	`if (Debug .infoOn ()) Debug .logInfo ("The TrackingCode  with  ID [" + trackingCodeId  + "] has expired, ignoring this trackingCodeId" , module );`

4.3 File Organization

4.3.1 Blank lines and optional comments are used to separate sections.

Lines 200-201 are both blank, but 1 is enough.
Lines 207, 438, 474 are blank but don't separate any section.

4.3.2 Where practical, line length does not exceed 80 characters

Line	#char
107	111
124	117
145	116
149	93
179	110
183	109
191	86
192	102
193	93
197	83
216	113
221	105
226	119
228	108
230	84
236	116
238	105
240	83

Line	#char
245	84
261	101
266	85
272	95
277	88
297	117
305	105
308	110
313	112
330	109
345	98
346	114
347	90
349	99
352	95
353	99
363	109
368	96

Line	#char
390	85
404	120
408	112
411	108
415	113
420	115
434	90
455	93
457	99
459	117
464	104
486	114
489	113
492	86
493	97
494	98
495	99
497	81

4.3.3 When line length must exceed 80 characters, it doesn't exceed 120 characters.

Line	#char
100	139
102	143
111	121
116	143
118	188
134	173
154	138
156	160
161	169
173	153
180	167
184	150
203	138
211	132
224	204
227	148
234	204
237	147
244	121
262	159
268	159
269	131
291	148
322	151
234	163
329	153
335	126
336	183
339	125
340	166
344	126
384	139
386	151
392	121
393	126
398	150
399	124
403	163
407	140
433	186
465	136
479	135
481	147
487	171
490	154
502	141

4.4 Wrapping Lines

4.4.1 A new statement is aligned with the beginning of the expression at the same level as the previous line.

The following lines have a wrong indentation:

267	`if (cookieDomain .length() > 0) siteIdCookie .setDomain(cookieDomain);`
	`    response .addCookie(siteIdCookie);`

273	`if (cookieDomain .length() > 0) updatedTimeStampCookie .setDomain(cookieDomain);`
	`    response .addCookie(updatedTimeStampCookie );`

4.5 Comments

4.5.1 Comments are used to adequately explain what the class, interface, methods, and blocks of code are doing.

Line 124:

```
//null out userLogin fields, no use tracking to customer, or is there?; set dates to current
```

Comment is not comprehensible, it seems like a note forgotten by the developers.

Lines 268, 262:

```
// if trackingCode.siteId  is  not null  write a trackable cookie with name in the form:
   Ofbiz.TKCSiteId  and timeout will be 60 * 60 * 24 * 365
```

Comment at line 268 is the same at line 262, it's highly probable that the developers confused `Ofbiz.TKCSiteId` with `Ofbiz.TKCD.UpdatedTimeStamp`.

Line 330:

```
//this return value will be ignored, but we'll designate this as an error anyway
```

Line 353:

```
//don't return error, want to get as many as possible: return "error";
```

It's not clear from comments at line 330, 353 what's the intention of this block of code.

4.6 Java Source Files

4.6.1 Check that the javadoc is complete.

The following methods have incomplete Javadoc because parameters explanation is missing:

- **checkTrackingCodeUrlParam** (line 54)
- **checkPartnerTrackingCodeUrlParam** (line 92)
- **checkTrackingCodeCookies** (line 308)
- **removeAccesTrackingCodeCookie** (line 420)
- **makeTrackingCodeOrders** (line 434)

The following methods have no Javadoc at all:

- **ProcessTrackingCode** (line 173)
- **checkAccessTrackingCode** (line 363)

It would be better also if external objects like Delegator and VisitHandler had a better Javadoc, to fully understand why they are used in this class.

4.7 Class and Interface Declarations

4.7.1 Check that the code is free of duplicates, long methods, big classes, breaking encapsulation, as well as if coupling and cohesion are adequate.

The method **checkPartnerTrackingCodeUrlParam** is too long and it should be divided into at least three methods in order to provide more readability. The fields of empty tracking code (lines 122-128) and of the default tracking code (lines 142-156) should be put in separate methods.

Lines 122-128:

```
defaultTrackingCode .set("trackingCodeId" , trackingCodeId );
defaultTrackingCode .set("trackingCodeTypeId" , "PARTNER_MGD" );
//null out userLogin fields, no use tracking to customer, or is there?; set dates to current
defaultTrackingCode .set("createdDate" , UtilDateTime .nowTimestamp ());
defaultTrackingCode .set("createdByUserLogin" , null);
defaultTrackingCode .set("lastModifiedDate" , UtilDateTime .nowTimestamp ());
defaultTrackingCode .set("lastModifiedByUserLogin" , null);
```

Lines 142-156:

```
trackingCode  = delegator .makeValue ("TrackingCode" );
trackingCode .set("trackingCodeId" , trackingCodeId );
trackingCode .set("trackingCodeTypeId" , "PARTNER_MGD" );
//leave userLogin fields empty, no use tracking to customer, or is there?; set dates to current
trackingCode .set("createdDate" , UtilDateTime .nowTimestamp ());
trackingCode .set("lastModifiedDate" , UtilDateTime .nowTimestamp ());
//use nearly unlimited trackable lifetime: 10 billion seconds, 310 years
trackingCode .set("trackableLifetime" , Long.valueOf(10000000000L ));
//use 2592000 seconds as billable lifetime: equals 1 month
trackingCode .set("billableLifetime" , Long.valueOf(2592000));
```

The method **processTrackingCode** consists in 130 lines of code, definitely too many, and since three main functionalities has been identified in the functional role of this method, it could be divided in three sub methods.

The blocks between lines 444-450 and 252-259 are two different ways of executing the same operation, they can be uniformed and a private method can be created to execute this operation.

Lines 444-450:

```
 if (cookies != null && cookies.length > 0) {
     for (int i = 0; i < cookies.length; i++) {
         String cookieName = cookies[i].getName();
         // find the siteId cookie if it exists
         if ("Ofbiz.TKCD.SiteId".equals(cookieName)) {
             siteId = cookies[i].getValue();
             }
```

Lines 252-259:

```
String visitorSiteIdCookieName = "Ofbiz.TKCD.SiteId";
[...]
if (cookies != null) {
    for (int i = 0; i < cookies.length; i++) {
        if (cookies[i].getName().equals(visitorSiteIdCookieName)) {
            visitorSiteId = cookies[i].getValue();
            break;
            }
        }
    }
```

4.8 Initialization and Declarations

4.8.1 Variables are initialized where they are declared, unless dependent upon a computation.

Lines 61, 98, 320 : `GenericValue trackingCode;`

These lines declare a non initialized trackingCode; the code would be more readable if they were initialized to null.

4.8.2 Declarations appear at the beginning of blocks. The exception is a variable can be declared in a for loop.

There are many declaration which don't appear at the beginning of block:

- Line 269 should appear before 264.
- Lines 206, 208, 225, 235, 246, 278, 279, 282, 285, 292 should appear before 188
- Line 477 should appear before line 444
- Lines 492-495 should appear before line 485

4.9 Output Format

4.9.1 Check that the output is formatted correctly in terms of line stepping and spacing.

Lines 398, 403, 487, 490, 502 contain a "TrackingCode" string, while at line 407 it is reported as "Tracking code", they should be uniformed.

At line 269 a comma is written with a space before it, instead of after it.

5 Other problems

- At line 251 the type is declared as "`javax.servlet.http.Cookie[]`" but using just "`Cookie[]`" would be enough and would be more readable.
- The method checkTrackingCodeCookies always returns *"success"*; since that's the only possible string that can be returned this method could return *void* instead of *String*.
- A literal string is inside a comparison (equals) at line 455, but usually it's preferred to call the equals method on the literal String with the variable String as parameter.
- In checkAccessTrackingCode the *response* parameter is not used.
- The *makeTrackingCodeOrders* returns a list of *GenericValue* that cannot have more than one element. If the developers wanted to create a list of more than one *GenericValues* the lines 477-498 should have been put inside a *for* cycle, otherwise it's useless to create and return a List. Here follows the block of code mentioned, with the most significant statements highlighted:

```
List<GenericValue > trackingCodeOrders  = new LinkedList<GenericValue >();
[...others lines in which trackingCodeOrders  isn't used...]
if (trackingCode  != null) {
   //check effective dates
   if (trackingCode .get("fromDate" ) != null  &&
           nowStamp .before(trackingCode .getTimestamp ("fromDate" ))) {
       if (Debug.infoOn()) Debug.logInfo("The TrackingCode  with ID [" + trackingCodeId  + "] has
               not yet gone into effect, ignoring this trackingCodeId" , module);
   }
   if (trackingCode .get("thruDate" ) != null  &&
           nowStamp .after(trackingCode .getTimestamp ("thruDate" ))) {
       if (Debug.infoOn()) Debug.logInfo("The TrackingCode  with ID [" + trackingCodeId  + "] has
               expired, ignoring this trackingCodeId" , module);
   }
   GenericValue  trackingCodeOrder  = delegator.makeValue ("TrackingCodeOrder",
           UtilMisc .toMap("trackingCodeTypeId" , trackingCode .get("trackingCodeTypeId" ),
           "trackingCodeId" , trackingCodeId , "isBillable", isBillable, "siteId", siteId,
           "hasExported" , "N", "affiliateReferredTimeStamp" ,affiliateReferredTimeStamp ));

   Debug.logInfo(" trackingCodeOrder  is " + trackingCodeOrder , module);
   trackingCodeOrders .add(trackingCodeOrder );
   } else {
       // Only log an error if there was a trackingCodeId  to begin with
       if (trackingCodeId  != null) {
           Debug.logError("TrackingCode  not found for trackingCodeId  [" + trackingCodeId  + "],
               ignoring this trackingCodeId." , module);
           }
       }

return trackingCodeOrders ;
```

6 Hours of work

6.1 Luca Oppedisano

21/01/2017	2h00'
22/01/2017	7h30'
25/01/2017	3h00'
26/01/2017	1h30'
30/01/2017	0h30'
02/02/2017	0h30'
04/02/2017	0h30'

6.2 Davide Moneta

22/01/2017	3h30'
23/01/2017	3h30'
24/01/2017	1h15'
25/01/2017	3h00'
26/01/2017	1h00'
02/02/2017	0h30'

6.3 Federico Oldani

21/01/2017	1h30'
22/01/2017	2h00'
24/01/2017	1h00'
25/01/2017	2h00'
26/01/2017	1h30'
01/02/2017	1h00'